upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa

TFG

Mikel Salinas López

SÍNTESIS DE IMAGEN A PARTIR DE AUDIO

Sistema modular capaz de generar vídeo a partir de información e interpretación de una pieza musical.

Tutor: Rafael Cabeza
Departamento de ING. ELECTRICA Y ELECTRONICA

CONTENIDOS

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCIÓN

Motivación

En primer lugar, se comentan los principales intereses que han motivado la realización del proyecto 'Síntesis de vídeo a partir de audio', en muchos sentidos, comunes a los que orientaron mi decisión a cursar el grado en tecnologías de la Comunicación.

Por un lado, la idea de acercarme de forma técnica al mundo de lo musical, de forma que llegue a entender algunas de las propiedades físicas que lo caracterizan. Es en este contexto en el que decido guiar mi trabajo fin de grado hacia el tratamiento de señales de audio.

Durante el grado, además de explotar este interés, y como es normal, encuentro otros campos que me son sumamente interesantes. Concretamente, las asignaturas relacionadas con el tratamiento de imagen me dieron a conocer un aspecto, a mi entender, interesante y estético.

Con el pretexto de aunar estos dos intereses, me propongo buscar un proyecto capaz de contener aspectos de los dos campos. La primera idea que consigue dicho propósito es la de diseñar un sistema productor de vídeo en streaming. La señal de entrada del mismo sería un audio interpretado en el momento, el sistema tendría una serie de patrones gráficos que iría cambiando en consonancia con el bpm de audio introducido, así como de otras características detectables, como la intensidad sonora o la presencia de diferentes rangos de frecuencia.

El sistema tendría que resultar de ejecución rápida y capaz de adaptarse a cambios en el audio. Como me aclaro el tutor, en este supuesto sistema el énfasis se colocaba en la parte de imagen, siendo el resultado muy subjetivo. Es decir, a la hora de evaluar el resultado obtenido, no quedaría claro el trabajo invertido. Para poder evaluar con criterio el trabajo, debería establecerse una correlación más fuerte entre audio y vídeo. Algo que sea fácilmente perceptible.

Rafael me propuso, relacionado con las ideas que le plantee en un inicio, un proyecto que sin duda me resulto más adecuado a lo citado. La propuesta era empezar el diseño de un sistema capaz de relacionar una interpretación de una pieza musical clásica con una representación que resultase natural.

La idea cobro forma al ver el trabajo realizado por un compositor de música contemporáneo, de nombre Stephen Malinowski. Sus representaciones a partir de interpretaciones de obras clásicas, aunque presentan ciertas semejanzas con algunas representaciones midi convencionales, resultan más naturales y expresivas. Al usar como audio una interpretación real, en lugar de una síntesis a partir de midi, están presentes todas las figuras del lenguaje musical.

Malinowski no sólo sincroniza cada nota con el momento real de interpretación, sino que tiene en cuenta esas figuras, así como la variabilidad natural característica de una interpretación humana de la música.

En sus representaciones se puede observar, por ejemplo, la variación de intensidad sonora en cada parte, así como desviaciones de frecuencia (vibratos), entre otros.

El propósito del proyecto será crear la base de un sistema capaz de realizar logros similares.

Descripción del problema

Una vez descrita la meta del proyecto, se presta atención al problema técnico que conlleva. Éste radica en el análisis de pistas de audio, así como la lectura e interpretación de información sobre las mismas.

El foco principal de atención será la sincronización entre información codificada e interpretación de una pieza musical. Como es de suponer, la información que podamos encontrar sobre una pieza puede ser de diferente índole. Es muy común, hoy en día, el trabajo con versiones digitales de partituras, de diferente formato y uso. Como es de suponer, siempre habrá una diferencia entre los tiempos y notas expresados en una partitura, ya sea digital o no, y la interpretación de una pieza musical.

Nuestro trabajo será obtener información exacta de tiempos y notas partiendo de dicha información. Para ello se recurrirá al filtrado por bandas de la señal de entrada, y la búsqueda a partir de la energía de los diferentes periodos temporales. El trabajo involucra, por tanto, la creación de bancos de filtro, así como de su correcto uso. Así mismo, se debe llevar un estricto control del tiempo, con el objetivo de localizar correctamente cada una de las notas interpretadas.

Además, el sistema debe ser capaz de reconocer archivos, posibles incompatibilidades, analizarlos y generar un vídeo en función de lo anterior.

Se considera este proyecto como base para un futuro sistema de carácter más complejo, por ello, no se prestará atención a toda la información sobre pista de audio que podamos encontrar en los diferentes formatos existentes. También se simplificará la representación final.

Metodología

Uno de los principales objetivos del trabajo es que resulte una buena base para un sistema más complejo y completo. Al tratarse de obtener una representación visual como salida del sistema, y dada la gran variedad de obras a introducir en el mismo, se considera que se pueden incluir incontables mejoras a partir de la base creada. Para ello, es necesario que el código que lo forme sea claro y ordenado, así como de fácil manipulación o sustitución.

Para cumplir estos requisitos, el sistema se dividirá en diferentes módulos correspondientes a las diferentes funcionalidades del mismo. De esta forma se consigue mayor maniobrabilidad a la hora de modificar o sustituir una de las partes.

Se irán describiendo los diferentes módulos por separado, y comentando sus puntos fuertes, así como los principales caracteres a mejorar en cada uno de ellos. También se consideran relevantes las diferentes dificultades a la hora de crear cada módulo, es por ello que se les otorgará gran parte de las explicaciones.

Estado del arte

Resulta complicado establecer los precedentes relacionados con este campo, ya que es bastante subjetivo y no resulta de gran interés comercial. Igualmente, cabe separar la materia relacionada con el proyecto en dos grandes bloques. Por un lado, más técnico, y motivo de estudio en diferentes ámbitos, se hablará de la sincronización y búsqueda de notas en señales de audio.

Se pueden separar este tipo de estudio por la forma de abordar el problema. Uno de los planteamientos utiliza únicamente técnicas de procesado; de esta forma, tratan de encontrar determinadas condiciones físicas que caracterizan el inicio de una nota. Dichas condiciones pueden estar relacionadas tanto con la energía de la señal como su fase en el momento de inicio de una determinada nota.

Generalmente el estudio de cambios en la energía de la señal es más útil en la detección de comienzos bien definidos, producidos en la mayor parte de las ocasiones por instrumentos de percusión. Por el contrario, los cambios en el dominio de la fase son sensibles a inicios de nota suaves, que son consecuencia del comienzo de una excitación, pero pueden no indicar un cambio en la energía de la señal. Se pueden encontrar estudios que proponen la combinación de ambos métodos. Este sistema dual suele mejorar los resultados de los métodos aplicados por separado, dado, en parte, a la gran variabilidad característica del entorno musical.

El otro planteamiento de estudio se basa en técnicas de aprendizaje automáticas. Éstas ayudan a establecer mejores fronteras de decisión entre fragmentos de audio que contienen inicios de nota y fragmentos que no los contienen [2]. Normalmente éstas se aplican sobre un dominio de dimensiones superiores.

Ambos planteamientos están ampliamente desarrollados para entornos offline, siendo materia de estudio actual la implementación de ellos, o variaciones de los mismos en entornos online, de alto interés hoy en día. Por el momento, esta materia no concierne al proyecto que describiremos a continuación, aunque no se puede descartar una futura migración a formatos online.

Se pueden encontrar tanto tesis como estudios variados sobre localización de frecuencias específicas en archivos de audio. Uno de los campos donde nos podemos encontrar este tipo de análisis es la detección de BPM (Beats Per Minute). Éste es un problema clásico en análisis de audio, y resultando de uso comercial, se ha desarrollado con cierta constancia en los últimos años.

Como factor común en la mayoría de estos estudios podemos encontrar similares estructuras de obtención de on-sets (comienzos de nota). Por un lado, se establece un banco de filtros centrados en las frecuencias correspondientes a las notas buscadas, suficientemente selectivos como para eliminar las notas de altura más cercana. Y por otro lado se realiza algún procesado para obtener la envolvente de la señal o de su energía, en algunos casos de su derivada o la de su fase, como se ha comentado anteriormente.

El otro ámbito de interés en referencia al proyecto son las diferentes representaciones que se pueden encontrar relacionadas con pistas musicales. Esta es la parte más subjetiva del proyecto, y aunque no resulta ni mucho menos despreciable, resulta más difícil sembrar precedentes sobre la misma.

Existen varios tipos de representaciones relacionadas con la música, y con diferentes objetivos.

Como añadido de algunos reproductores, así como de algunos editores de audio, se suelen encontrar representaciones tiempo-frecuencia con motivo visual, así como analítico. Sin embargo, este tipo de representaciones no logran generar sensación de fluidez musical. En la figura 1 se muestran algunos ejemplos.

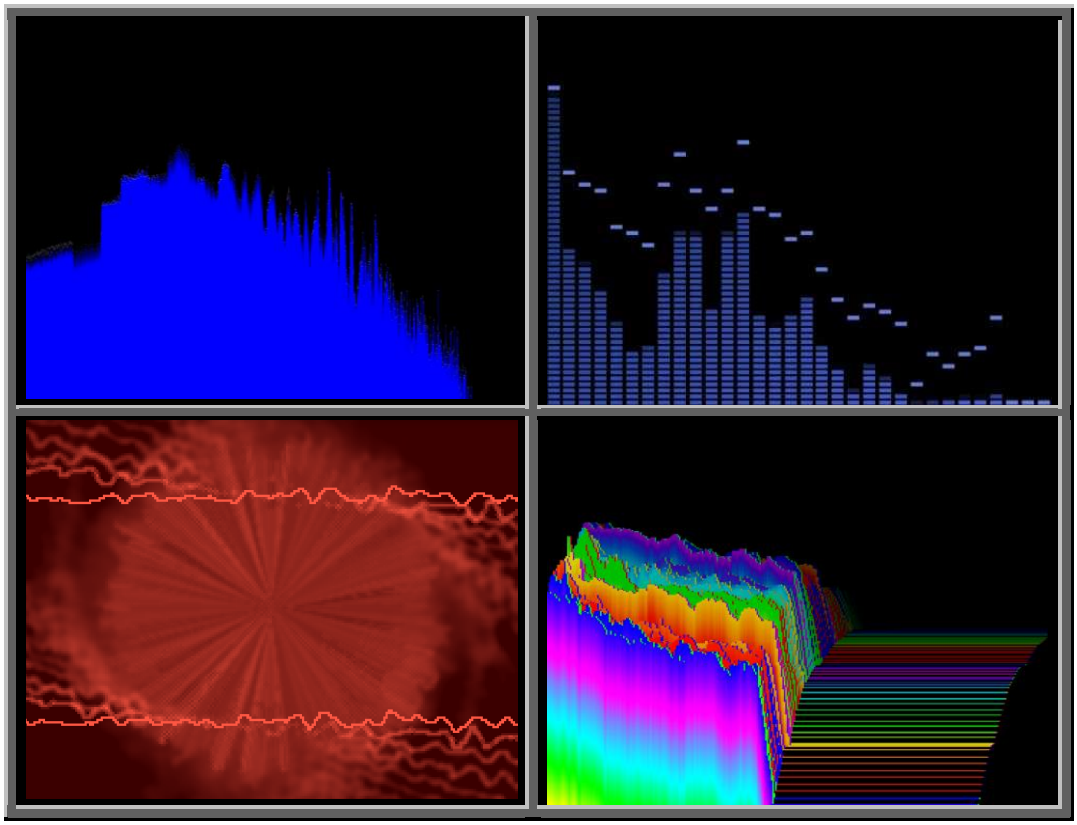


Figura 1

Por otro lado, se pueden encontrar representaciones directamente relacionadas con las notas presentes en la pieza una por una. En general éstas se basan en lectura de archivos MIDI, de los que hablaremos más adelante. Aunque éstas tienes en cuenta la altura de sus notas y su duración, suelen carecer de naturalidad musical, al no tener en cuenta las muchas propiedades musicales que dan forma a una pieza musical. En las figuras 2 y 3 vemos representaciones de este tipo.

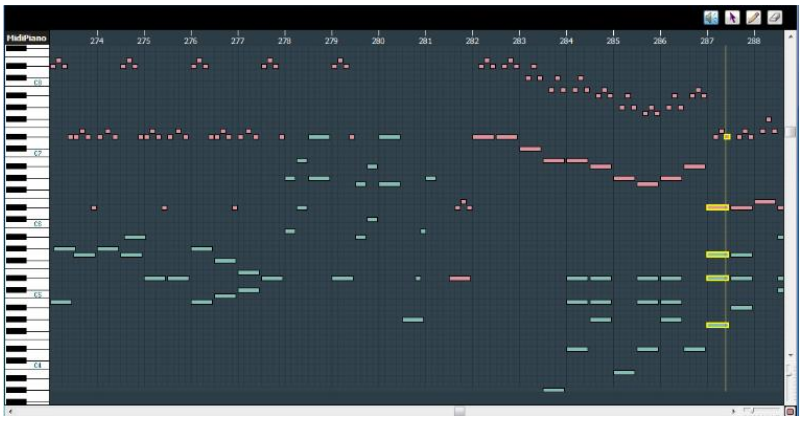


Figura 2

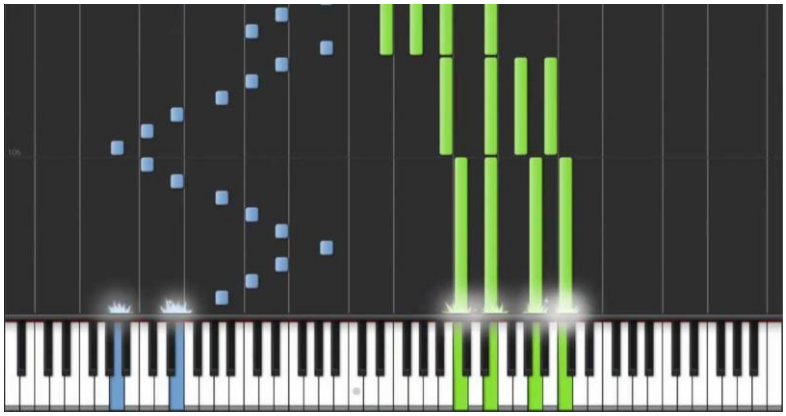


Figura 3

Es muy usual encontrar este tipo de representación en software de aprendizaje de piano, dada su estrecha relación.

Se encuentran, en menor medida, representaciones alternativas más naturales. Estas poseen características que permiten tener en cuenta una mayor parte de los recursos musicales. Pondremos como ejemplo, de aquí en adelante, el trabajo realizado por Stephen **Malinowski**. Compositor y desarrollador de software de origen americano mencionado anteriormente. Éste logra, mediante diferentes métodos, representaciones que retratan con eficacia distintos aspectos de la música, en su mayoría clásica.

En la figura 4 se puede observar un fragmento de uno de sus vídeos. En este caso, se trata de una pieza compuesta para oboe. Aunque se puede intuir la relación con una representación MIDI, resulta mucho más natural ya que se representa la unión entre las sucesivas notas, así como su variación en amplitud y frecuencia.

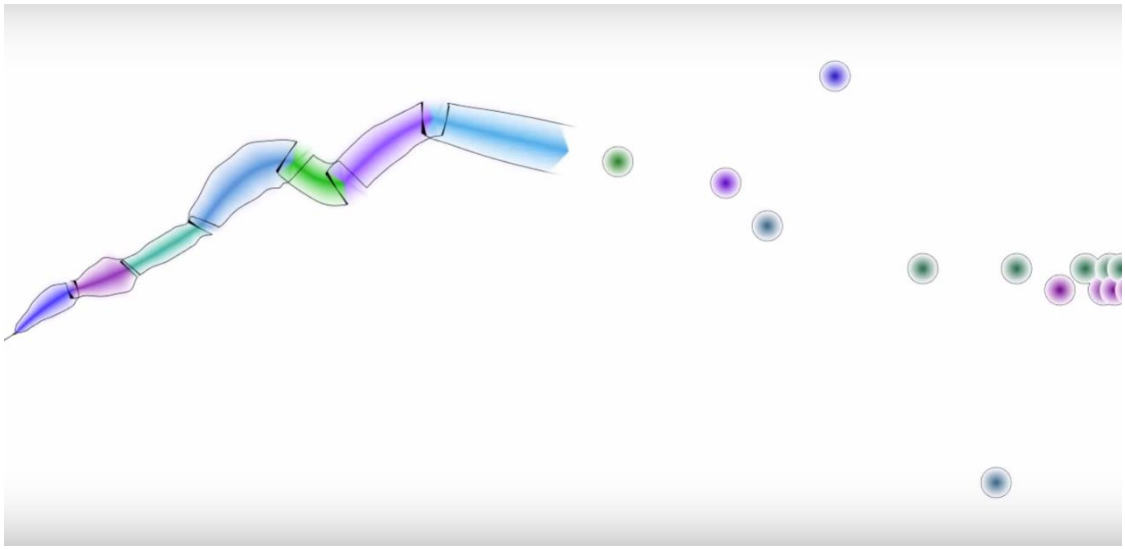


Figura 4

En la siguiente figura se observa otro tipo de representación, en este caso polifónica, ya que se trata de una pieza compuesta para piano.

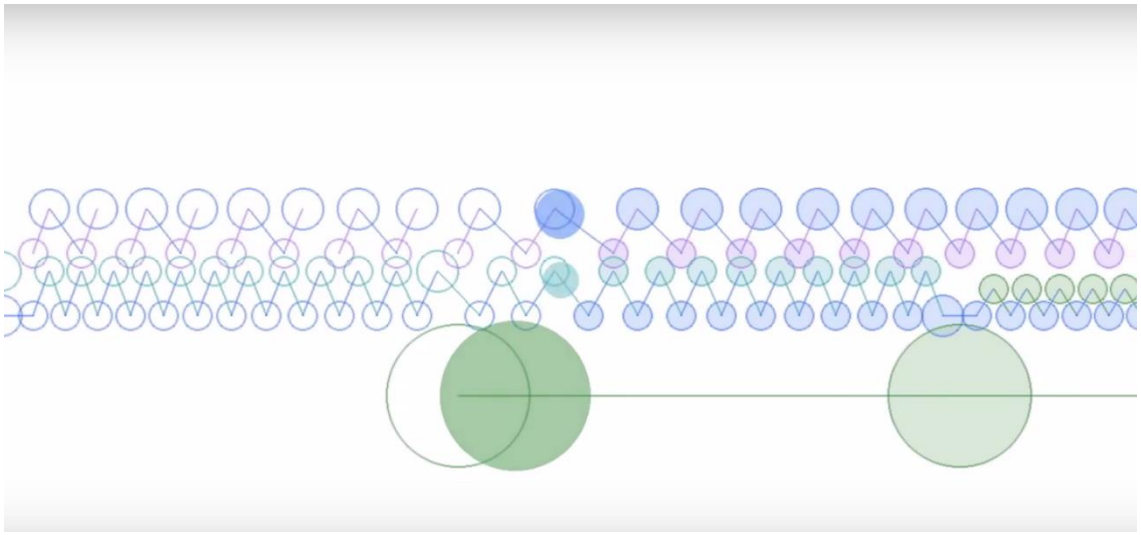


Figura 5

El mismo **Malinowski** explica de manera general los diferentes procedimientos seguidos para sincronizar el audio y el vídeo. En ellos, siempre es necesaria una parte manual de reajuste. Según su experiencia, el software se queda a medias al realizar la sincronización, y para un resultado perfecto, es necesario realizar ajustes manuales una vez éste ha terminado.

En este proyecto trataremos la parte de software en primer lugar, intentando que éste genere un resultado lo mejor posible. Si es necesario o no un reajuste manual una vez aplicado, será algo que se decidirá como una de las conclusiones de este proyecto.

Módulos del proyecto

En la siguiente parte de la memoria se describirán los diferentes módulos que componen el proyecto. Para cada uno de ellos se especificarán tanto sus funcionalidades dentro del sistema como su estructura y características principales. Además, se describirá el proceso seguido en su creación, los diferentes obstáculos encontrados y las posibles mejoras que se consideran más importantes a la hora de complicar el sistema.

PARSER

Descripción y funcionalidades

Se trata del primer módulo del sistema, se encarga de recibir los archivos y almacenarlos para su manipulación en Matlab. Así mismo, realiza un análisis simple de los archivos introducidos, y da una predicción de compatibilidad. Esto nos permite localizar fallos al introducir archivos.

Entradas

La función que constituye este módulo no tiene como entrada ninguna variable previa, sin embargo, permite la introducción de determinados archivos externos a Matlab.

La primera característica a definir previo a la construcción de este módulo es la compatibilidad del programa. Es decir, los archivos que permitirá introducir y, por tanto, será capaz de leer y utilizar.

A la hora de plantear el problema, surgen principalmente dos propiedades que han de tener los archivos a leer. Por un lado, se busca que el archivo a introducir resulte relativamente fácil de encontrar para diferentes y variadas piezas musicales. Por otro, y de igual importancia, se busca maximizar la información contenida en dichos archivos; esto nos permitirá disponer de más datos a la hora de analizar y sincronizar cada pista.

Es complicado encontrar un formato que ofrezca un compromiso entre estas dos cualidades. Dada esta dificultad, nos decantamos por hacer que la compatibilidad sea dual, permitiendo, así, dos tipos de archivo. Cada uno de estos dos archivos tendrá como fuerte una de las dos condiciones.

Los de tipo MIDI cumplirán la función de ser comunes y fáciles de encontrar, quedando en todo caso escasos en referencia a la información que proporcionan. Por otro lado, los archivos tipo MXML (XML comprimidos), no resultan tan comunes o fáciles de encontrar para determinadas pistas musicales; no obstante, otorgan la misma información que una partitura en formato físico clásica. En definitiva, se obtienen muchos más datos sobre el desarrollo musical de la pista. En el anexo 2 se explican de forma más extensa las principales características de ambos formatos.

La idea es que el programa funcione correctamente con el mínimo de información, y que el restante sirva únicamente para mejorar los resultados obtenidos o para acelerar el proceso de obtención.

Una vez decidido que tipos de archivo se pueden introducir, pasamos a explicar el método de introducción de los mismos. Éste se basa en ventanas emergentes, una específica para cada uno de los archivos.

Por un lado, se introduce un archivo correspondiente al registro de notas y tiempos; como se ha comentado, admitirá varios formatos. Por otro lado, el archivo correspondiente a una grabación de la interpretación de la pieza, siempre en formato de audio, permitiendo diferentes extensiones.

Las ventanas se configuran para admitir únicamente los tipos de archivo que el programa es capaz de leer e interpretar mediante la función "uigetfile" de Matlab. Los formatos permitidos son:

- Información de pista: midi (*.mid) y MusicXML (*.mxl)
- Audio: *.mp3, *.wav y *.wma

Se mostrará la ventana hasta que el usuario escoja un archivo. Una vez introducidos los dos archivos, se pasa a procesarlos.

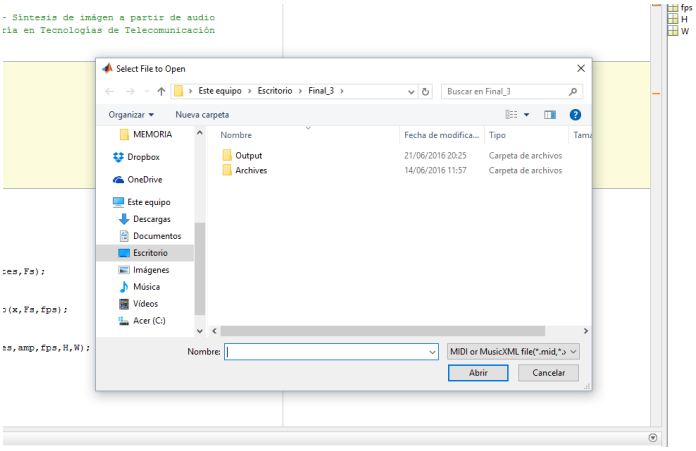


Figura 6

Procesado

En este apartado se explica y estudia el procesado realizado sobre los archivos introducidos, así como los pasos seguidos a la hora de crear el módulo.

Este módulo fue el primero que se creó y también el que menos cambios ha necesitado en el transcurso de creación del código. En concreto se formó en dos fases. En una primera se le atribuyen las funcionalidades básicas de introducción de archivos concretando formatos y extensiones de los mismos. Una vez creado, al utilizar el código, resultó pertinente modificarlo para evitar errores y mejorar la interacción con el usuario final. Se añaden las funcionalidades de reconocimiento de compatibilidad, así como la repetición de pantallas emergentes si es pertinente. Además, se añade una función para cortar la señal de audio en función del inicio y final de la música. Una vez hecha esta modificación, no se vuelve a actualizar el código.

Pasamos a explicar el funcionamiento del módulo. Para tratar los archivos, se almacenan en una variable tipo texto (string) las rutas concatenadas con sus nombres en variables. En el caso del archivo de información, se necesita conocer el formato escogido por el usuario. Para obtenerlo, se usa la variable tipo *string* almacenada, que contiene el nombre del archivo. Se separa la parte correspondiente al nombre de la correspondiente a la extensión, es decir, la que sigue al punto. Con este punto como elemento intermedio, y comparando este *string* con los dos posibles, se obtiene el determinado formato.

Para decodificar, en ambos casos se buscan paquetes de Matlab ya creados que simplifiquen el proceso. No resulta difícil encontrar trabajos previos de encriptación y des-encriptación de estos para la plataforma Matlab. Lo que si resulta un problema en ambos casos es la actualización de esto para compatibilizarlos con la versión utilizada de Matlab, así como con el procesamiento de 64 bits.

En las referencias ([3] y [4]) podemos encontrar los artículos relacionados con los scripts usados en el módulo. En ellos se explica tanto el contexto de utilización como el funcionamiento de dichas herramientas. Para ambos debemos hacer cambios relacionados con la compatibilidad, como ya se ha comentado.

Por un lado, en el caso del MIDI se necesita cambiar funciones, facilitadas por los creadores del toolbox. Una vez sustituidas el decodificador funciona correctamente.

En el caso del MusicXML hace falta volver a compilar ciertas funciones creadas en el lenguaje C, para que sean ejecutables en un sistema procesando a 64 bits. Una vez realizados los cambios, ambos tipos de archivo nos devuelven la información suficiente para sincronizar la imagen y la interpretación musical.

Como mejora sustancial se propone utilizar más datos en el caso del archivo MusicXML. Se comentará en apartados posteriores.

De aquí en adelante se explicará la implementación de funciones de ambos grupos de herramientas para la lectura de los dos tipos de archivo.

En el caso de que el archivo introducido sea MIDI, se utiliza la función 'midi2nmat'. Esta forma parte del toolbox llamado miditools; como ya se ha comentado, en las referencias queda indexada la fuente de los archivos utilizados, así como la explicación de su uso [3]. Mediante esta función se consigue transcribir el archivo a una variable tipo array de Matlab. Ésta tiene el siguiente formato:

Columna 1	**Columna 2**	**Columna 3**	**Columna 4**
Tiempos/beats	Duraciones	---	Notas (cod. MIDI)

Tabla 1

En el caso contrario, con archivo de tipo MusicXML, se usa la función llamada 'get_xml_notes', que está contenida en el paquete de lectura de XML.

Éste contiene todo tipo de funciones para des encriptar este lenguaje y guardar la información contenida en el archivo dentro de variables de Matlab. En nuestro caso usaremos únicamente una de las variables. Concretamente la llamada 'raw_notes', que nos da la información de inicio y duración de cada nota, así como la altura y octava de la misma. Se realizarán pequeñas transformaciones en el array para que la información aparezca de forma equivalente a la variable obtenida a partir del MIDI. De esta forma se simplifican los siguientes módulos.

Por otra parte, en referencia al archivo de audio, se utiliza la función 'audioread' de Matlab. La salida de esta función es una variable con las muestras de la señal de audio en estéreo, además de la frecuencia de muestreo de la pista.

El siguiente paso es buscar en esta variable el inicio y final exacto de la música, eliminando, si es necesario, silencios previos. Este procesado es importante para asemejar lo máximo posible los dos archivos, ya que el midi no tendrá silencios previos a las notas. Para realizar este procesado se crea la función "find_beginning".

find_beginning

Ésta trabaja con la energía de la señal, en concreto con sumas de muestras en intervalos de tiempo cortos. Se busca la aparición de la primera nota, es decir, un incremento de energía perceptible y significativo respecto al resto de señal. Es necesario trabajar con sumatorios ya que las señales de audio tienen naturaleza periódica, de esta forma atenuamos el efecto de los constantes cruces por cero.

Se itera la suma de muestras buscando este aumento significativo entre sucesivos intervalos. El diferencial usado como mínimo dependerá del máximo de la señal de audio. Será a partir de ese incremento desde el cual se tome las muestras de la señal para tratar. La función se ha ajustado experimentalmente con varias piezas musicales de diferentes estilos.

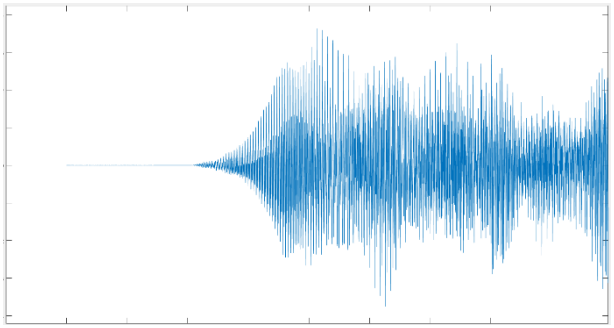


Figura 7

Aquí podemos observar uno de los dos canales de una señal introducida. Se puede ver fácilmente un silencio al principio de la misma; éste podría desajustar la sincronización si no se elimina o se tiene en cuenta.

Al elevar la señal al cuadrado tenemos la energía de la misma.

Al aplicar el algoritmo buscamos un incremento mayor al máximo hallado dividido entre 150. De forma experimental resulta ser buena aproximación al sonido perceptible.

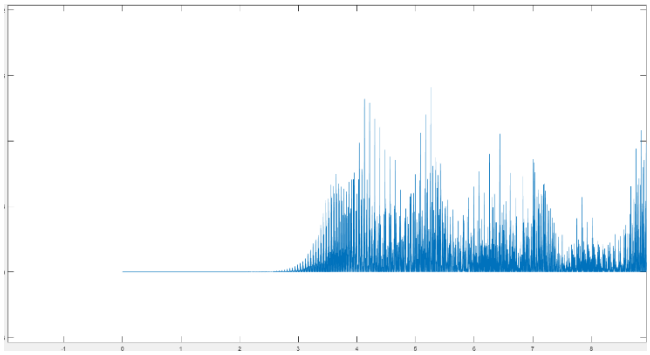


Figura 8

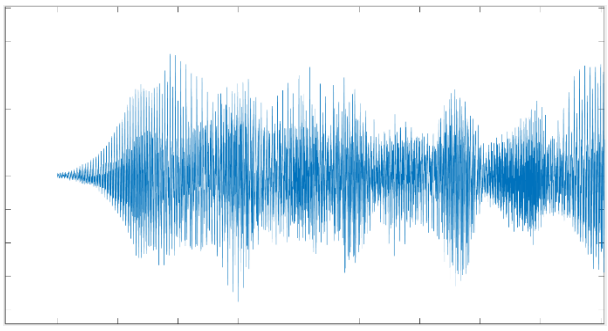


Figura 9

Una vez aplicado el algoritmo, se recorta en función de la primera muestra obtenida, quedando la siguiente forma de onda.

El proceso se repite para el final de la señal. Los pasos son iguales, pero se aplican sobre la señal invertida, de forma que se encuentra el final del sonido.

Una vez tenemos tanto el audio como la información lista para procesar, pasaremos a comprobar que los archivos introducidos son compatibles. Se usan dos métodos

combinados para decidir si los archivos introducidos pertenecen a la misma pieza musical. Por un lado, se hace un sencillo estudio sobre las notas presentes en ambos.

Se toma como referencia el archivo de información y se guarda una lista con todas las notas que deben aparecer en el audio. Así, se crea un vector con la columna correspondiente a las notas en MIDI presentes en la pista. Este vector hay que depurarlo, es decir, se deben eliminar las muchas redundancias presentes, ya que habrá muchas repeticiones de notas. Para ello usamos la función "delete_repeat".

El procedimiento que sigue es; posicionarse en el valor con índice uno, lo toma como referencia y recorre el resto de valores buscando coincidencias. Si la encuentra elimina el determinado valor del vector, y guarda la nueva longitud, para evitar problemas de índices fuera de rango de valores del vector.

De esta forma, se consigue un vector con las frecuencias en codificación MIDI que aparecen en el archivo. El siguiente paso es buscar las frecuencias en el audio. Es esta parte del procesamiento, se ha considerado más relevante la rapidez del algoritmo en contraposición de la exactitud, ya que sólo se necesita una estimación aproximada del parecido de ambos archivos. Una vez contextualizado el uso de esta parte del algoritmo, se explica su funcionamiento.

El algoritmo se basa en una búsqueda de valores altos en el espectro de la señal de audio. Una vez obtenido este mediante la función 'fft' nativa de Matlab, se recorren las frecuencias de las notas que deben estar presentes. En la figura 10 se puede ver el espectro de una pieza musical de prueba:

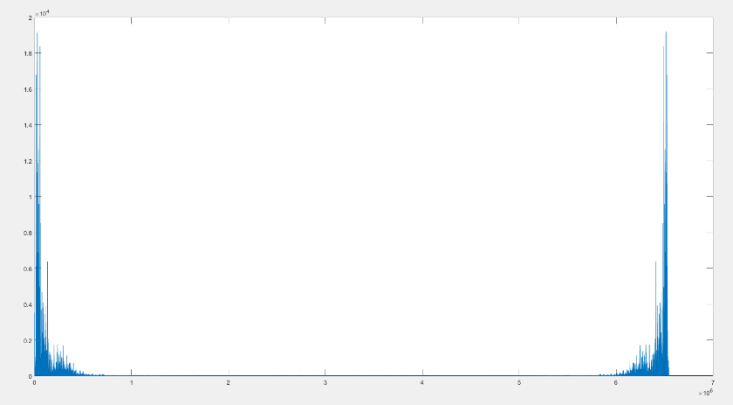


Figura 10

En la siguiente, figura 11, se observan, gracias al zoom, picos correspondientes a frecuencias presentes de forma continuada en la pieza.

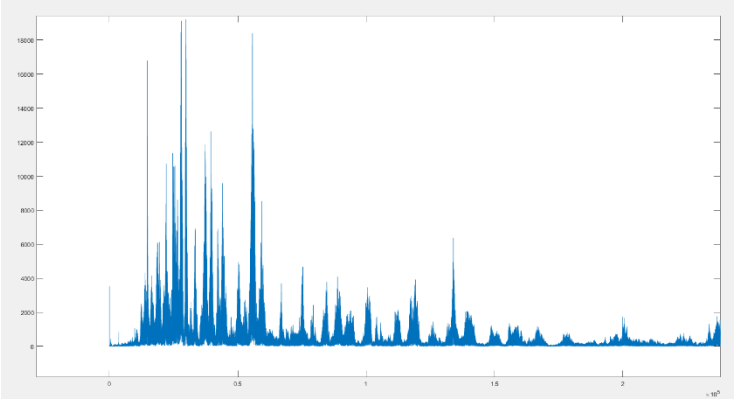


Figura 11

Para cada una de las notas se establece un rango de muestras. Estas representan las frecuencias cercanas a ella. De cada uno de estos conjuntos computamos el máximo y este se compara con un cuarto del máximo espectral total. En la siguiente figura vemos dicho umbral expresado como una línea de color rojo.

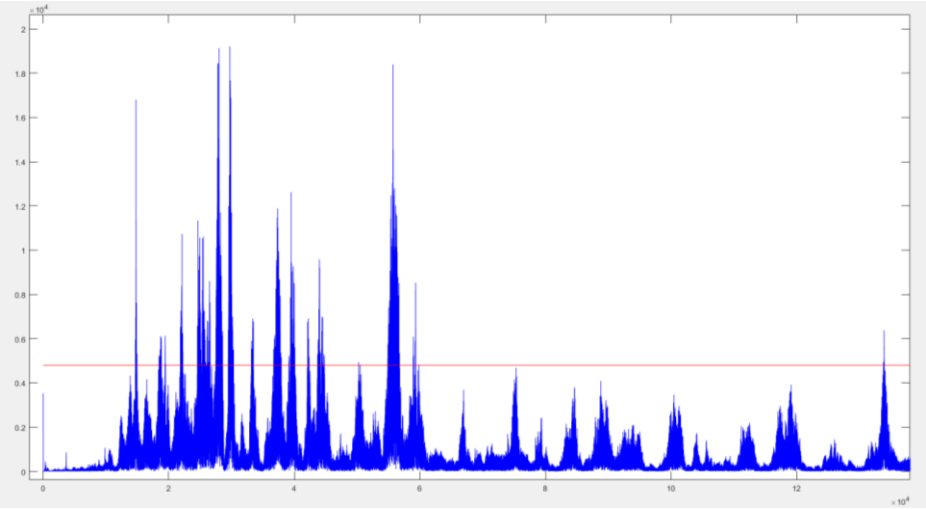


Figura 12

En el caso de que la diferencia entre ambos valores no sea muy grande, tomaremos la nota como incluida en el audio. En caso contrario, lo más probable es que no se encuentre esa nota en el mismo.

Una vez aplicado este proceso, se tiene el porcentaje de notas encontradas en el MIDI que, con alta probabilidad aparecerán en el audio. Se establece una ratio que distinga los archivos compatibles de los que no lo son. Se le da cierto margen al cálculo, relacionado con la probabilidad de error del método.

El otro procedimiento para determinar la similitud de los archivos introducidos es la longitud de los mismos. En el caso del audio, es sencillo establecer la duración de pista, dado que tenemos la información de la frecuencia de muestreo utilizada en su codificación. Se utiliza la longitud en muestras para, dividida entre esta frecuencia, obtener los segundos de la pista.

En el caso del MIDI se distinguen dos posibilidades. Que se detecte correctamente el BPS del tema, por tanto, se multiplican los beats finales del tema y el inverso del BPM obtenido. La otra posibilidad es que se detecten varios BPM's, y se pase a representar

en segundos directamente, en este caso se usa simplemente el último minutaje datado en el array.

Por último, al igual que en la anterior condición, se crea una ratio en función de la relación entre longitudes y se establece un mínimo de similitud dejando suficiente margen.

En el caso de que las dos ratios sean positivos, es decir, se cumpla el mínimo establecido, se aceptarán los archivos y se recibirá un mensaje de compatibilidad positiva desde la ventana de comandos. Si uno de las dos ratios resulta menor de lo establecido, se mostrará un mensaje de compatibilidad negativa y se muestra un mensaje por ventana de comandos. Se le da la opción al usuario de continuar con el proceso. En caso de dar respuesta negativa, se vuelven a mostrar las pestañas de selección de archivos.

Salidas

El script tendrá tres variables como salida:

- Vector con las muestras de la señal de audio.
- Frecuencia de muestreo de la señal de audio.
- Array con información de inicio, duración y frecuencia de las notas.

Mejoras

Como futura mejora del sistema, sería conveniente aprovechar toda la información obtenida gracias al archivo MusicXML, ya que este contiene todas las anotaciones musicales propias de una partitura.

Esta información extra puede servir, por ejemplo, para tener cada compás perfectamente situado. Además, podemos tener en cuenta todos los recursos musicales que tengan efecto sobre la intensidad, frecuencia o duración de la nota.

Como resultado, será posible mejorar o acelerar el proceso de sincronismo, así como ayudar al algoritmo de representación a obtener mayor veracidad respecto al audio.

SINCRONÍA

Descripción y funcionamiento

Este segundo módulo del proyecto será el encargado de sincronizar los archivos, que habrán sido previamente introducidos y guardados en variables de Matlab.

Éste ha resultado el módulo más conflictivo, como ya se mencionaba previamente. El código que se atribuye al resultado final ha pasado por muchos cambios y procesos de depuración.

Entradas

La entrada del módulo será, por un lado, la señal de audio en muestras, así como su frecuencia de muestreo. Por otro lado, tendremos el array con la información del inicio, duración y frecuencia de las notas presentes.

En futuras implementaciones mejoradas se incluirán como entradas a esta función más variables propias de la partitura digital de la pieza. Esto ayudará a mejorar la sincronización.

Filtrado

Antes de tratar el procesamiento que realiza este algoritmo, es importante hablar del problema general de creación de un banco de filtros y el sucesivo filtrado. Este tema es el que más involucra teoría de señal, por ello se le dedica un apartado específico.

Como es de suponer, se va a necesitar un banco de filtros correspondiente a las frecuencias de las notas que aparecen en la pieza musical. Estos filtros, creados para las determinadas frecuencias, tendrán un ancho de banda de un doceavo de octava. Es decir, las frecuencias de paso serán las correspondientes a cada nota de la escala musical, y con ellas buscaremos las de corte y paso para que el filtro rechaza las frecuencias de notas consecutivas. Las de paso serán frecuencias entre la nota buscada y la siguiente, concretamente, se hallarán dividiendo la octava en veinticuatro partes, sería equivalente a un cuarto de tono. Matemáticamente esto se expresa de la siguiente manera.

$$f_{pass_1} = \frac{f_c}{K'}$$

$$f_{pass_2} = f_c * K'$$

$$con \quad K' = 2^{1/24}$$

En el anexo 5 (Teoría musical), se explica de donde proviene esta relación, además, se dan algunos conceptos generales para contextualizar la matemática relacionada con las escalas musicales. Por otro lado, las de corte serán las frecuencias situadas previa y posteriormente, es decir:

$$f_{cut\_1} = f_c/K$$

$$f_{cut\_2} = f_c * K$$

De esta forma se asegura para las frecuencias contiguas sufrirán una atenuación considerada, al mismo tiempo que se toleran ciertas derivas en frecuencia por desafine del instrumento o interpretación libre.

Los filtros digitales se pueden clasificar principalmente en dos, **FIR**, e **IIR**. En cuanto a los FIR, se caracterizan por tener respuestas finitas, es decir, se trata de filtros no recursivos y estables. Esto es debido a estar compuestos únicamente por **ceros**. El inconveniente principal radica en su alto orden, así como su diseño más complejo para generar filtros selectivos. Las principales ventajas son su estabilidad y respuesta lineal en fase.

Por otro lado, los filtros tipo **IIR**, pueden presentan tanto **polos** como **ceros**, luego para entradas finitas, generan ilimitadas muestras no nulas a su salida. Se caracterizan por ser más fáciles de implementar, ya que necesitan menor orden para unas características dadas. Los filtros analógicos pueden ser transformados a su equivalente IIR, en contra de los **FIR**, que no tienen equivalente analógico.

En contraposición, aun cuando son diseñados para ser estables, hay posibilidad de que no resulten serlo. Por otro lado, no pueden diseñarse para tener una respuesta lineal en fase, aunque se pueden aplicar técnicas para lograrlo. Una de ellas es el filtrado bidireccional, que se comenta más adelante.

Las herramientas presentes en Matlab nos permiten crear los dos tipos de filtros, Debido a sus mejores características, se empieza creando bancos de filtros **FIR**. Dadas las características del filtro a diseñar, se prueban varios métodos. La razón para descartarlos es de índole computacional, ya que requieren de muchos más cálculos. Una vez probado, se decide recurrir a filtros **IIR**. Por definición, estos necesitan menos cálculos, orden y memoria, para las mismas especificaciones.

La aproximación aplicada para obtenerlos dependerá de las prioridades que se establezcan.

Para la correcta detección de las notas, se necesita que la respuesta del filtro sea lo más fiel posible.

Afectará, entre otros factores, el rizado del filtro en la banda pasante. No tanto el rizado en la banda de rechazo, ya que usualmente dicha banda no contendrá mucha energía. El rizado en pasante no sólo distorsiona la amplitud de la señal en parte de la nota, sino que puede llegar a generar detecciones erróneas de final de nota. Este caso se explicará con más detenimiento en sucesivos apartados.

Por otro lado, el transitorio del filtro tendrá un papel importante tanto en la amplitud como en el retardo introducido a la nota, siendo la localización de la misma, el factor más relevante a determinar.

De una forma menos directa, el orden necesario para aproximarse al ancho de banda y rizados que se especificarán, repercutirá en la velocidad de ejecución del algoritmo.

Conociendo esta dependencia entre tipo de filtro y resultado, se estudian los principales métodos de aproximación, así como sus ventajas y desventajas.

Aproximación Butterworth

La aproximación de Butterworth se obtiene al imponer que la respuesta en magnitud del filtro sea máximamente plana en las bandas pasante y de rechazo.

Presentan una respuesta en frecuencia monótona decreciente de pendiente proporcional al orden impuesto.

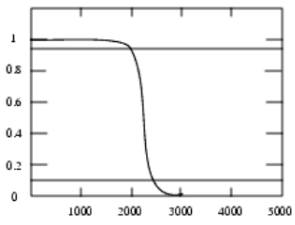


Figura 13

Aproximación Chebyshev

Figura 13

Los filtros de Chebyshev consiguen una caída más abrupta a frecuencias bajas en base a permitir un rizado de la respuesta en frecuencia en alguna de las bandas. Los hay de dos tipos:

- Tipo I
  - Son filtros solo polos
  - Presentan rizado constante en la banda pasante
  - Presentan una caída monótona en la banda no pasante.
  - Presenta una caída más abrupta que el Butterworth.

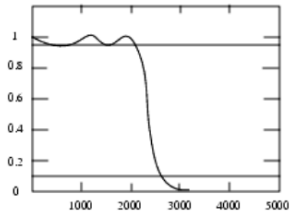


Figura 14

- Tipo II
  - Presentan ceros y polos
  - Presentan rizado constante en la banda no pasante
  - Presentan una caída monótona en la banda pasante

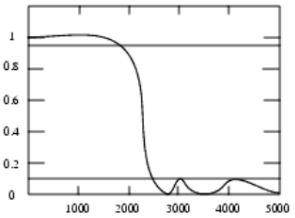


Figura 15

Aproximación Elíptica

Los filtros elípticos o de Cauer consiguen estrechar la zona de transición permitiendo un rizado constante en ambas bandas. Resultan los más eficientes, en tanto que obtienen el menor orden para una anchura de la banda de transición determinada.

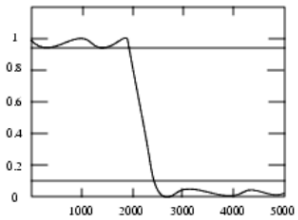


Figura 16

En resumen, la aproximación Chebyshev presenta mejores características que la Butterworth en la banda de transición. A altas frecuencias, en el rechazo de banda, ambas presentan un buen comportamiento, pero en el caso de utilizar el método Butterworth, serán necesarios órdenes significativamente más grandes.

Hay que tener en cuenta que en este tipo de bancos de filtros el factor de calidad debe ser constante con la frecuencia, es decir, en principio los órdenes de los filtros necesarios se mantienen. Esto se debe a la propiedad logarítmica de la escala musical.

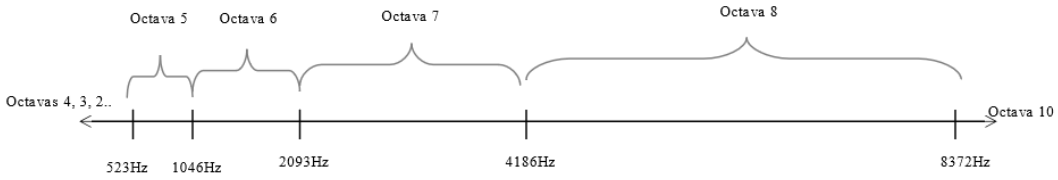


Figura 17

Se define el factor de calidad de un filtro como:

$$Q = \frac{f_c}{BW}$$

En la implementación, podemos observar que el ancho de banda de los filtros se calcula en función de la frecuencia central, siendo este:

$$BW = f_c * 1.06$$

Luego:

$$Q = \frac{f_c}{f_c * 1.06} = \frac{1}{1.06} = 0.9434$$

La aproximación elíptica es la que presenta un mejor comportamiento en este último sentido, al poseer una banda de transición más estrecha, comparativamente para un orden dado del filtro. No obstante, ésta presenta rizado constante en el paso y rechazo de banda. [6]

En principio, para nuestras necesidades, cabe pensar que el más apropiado es el método Chebyshev tipo 2, dada su banda pasante con rizado nulo y su caída abrupta para órdenes menores que el Butterworth.

Por otro lado, el tipo de filtrado a realizar, una vez se tiene el banco de filtros, también será relevante a la hora de determinar la localización de las notas.

En concreto, tendrán un papel importante las condiciones iniciales a la hora de filtrar. En una primera aproximación, se plantea el filtrado de la señal por partes, guardando las condiciones finales y utilizándolas en el segmento de señal consecutivo. Resulta obvio que utilizando este método será necesario el filtrado de toda la señal, aunque éste sea por partes.

Como alternativa a este tipo de filtrado convencional, se puede plantear un filtrado bidireccional de la señal. Éste se realiza filtrando de la señal en dos sentidos. La primera es un filtrado corriente, que produce una salida con cierto retardo dependiendo del filtro. Esta salida se invierte y se vuelva a filtrar, en la salida final, se compensan los retardos introducidos por el filtro. Esto es factible dada la condición de trabajar con señales guardadas.

Para sacar de la ecuación las condiciones iniciales, se realiza el filtrado sobre la señal con márgenes en inicio y final (conociendo el transitorio del filtro), y después se eliminan esos márgenes. Esto evita tener que filtrar toda la señal, de esta forma, se puede ir cogiendo las partes que nos interesen sin preocupación por posibles estados iniciales. Resulta más eficiente.

Procesamiento

Una vez introducidos los conceptos principales sobre el filtrado, pasamos a explicar el funcionamiento del módulo de sincronismo. Resulta el más complejo de todos, y contiene varias funciones anidadas. Según aparezcan, se dedicarán apartados para explicarlas con más detenimiento.

El primer paso que da el script es reorganizar la información de la variable con información sobre notas, cargada anteriormente. Ya se ha indicado previamente el formato de la misma; aquí queda representado:

Columna 1	Columna 2	Columna 3	Columna 4
Tiempos/beats	Duraciones	---	Notas (cod. MIDI)

Tabla 2

El cometido de este script es sincronizar esta variable con el audio introducido; para ello se irán cambiando los valores, así como guardando copias de los mismos y marcando con *flags* cuando sea necesario. Como consecuencia de estas necesidades, se reconfigura el formato de la variable, quedando de la siguiente forma:

Columna 1	Columna 2	Columna 3	Columna 4	Columna 5	Columna 6
Tiempos/beats 1	Duraciones 1	Tiempos/beats 2	Duraciones 2	Notas (cod. MIDI)	*Flag* detección

Tabla 3

Al desarrollar el script, se vio la necesidad de ir guardando dos versiones de inicios y duraciones. Por un lado, los valores que se van encontrando en el archivo, que no siempre serán fiables, y por otro los referenciados a los datos de MIDI o MXML. Más adelante veremos que estos se van recalculando en función de los primeros, siempre que se cumplan ciertas condiciones.

Las primeras pruebas con un solo vector de tiempos, resultaban muy susceptibles a errores del algoritmo de búsqueda.

Una vez reorganizado el array, el módulo pasa a inicializar los filtros necesarios, es decir, una serie de filtros centrados en las frecuencias que vayan a aparecer en el audio. Antes de nada, se establece este rango de notas a partir del MIDI o MXML. Para ello, se extrae el vector referente a frecuencias del array y se eliminan valores recurrentes. Se usa la función llamada 'delete_repeat', ya utilizada en el módulo previo.

Una vez se tiene el vector con las frecuencias, se ordena de forma ascendente y se le pasa a la función ***'init_filters'***. Ésta se ocupará de crear los filtros y extraer datos para mejorar la interacción con ellos. En el anexo 4 (funciones), se explican tanto esta función como las anidadas. La encargada de la creación de filtros es ***'create_filters'***.

El resultado de aplicar estas funciones al rango de notas extraído es un banco de filtros como el siguiente (figura 18):

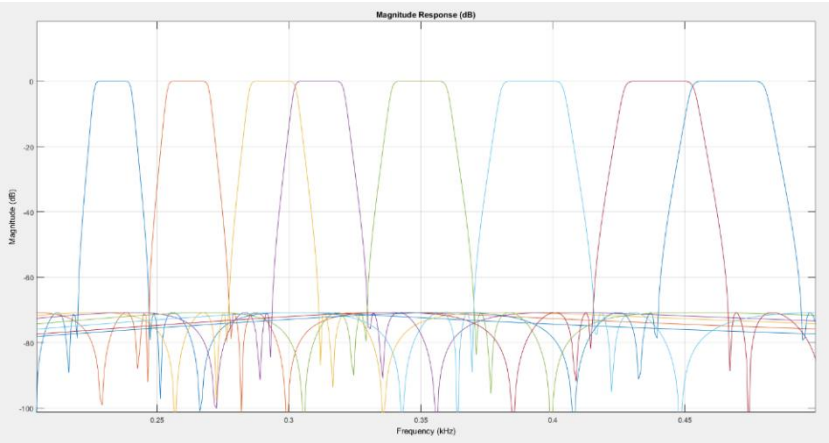


Figura 18

Como se puede observar, en este caso no están presentes en el rango todas las frecuencias de forma consecutiva, solo algunas de ellas. También queda representado, en el caso de notas adyacentes, el solapamiento entre filtros obtenido dados los límites para bandas pasante y de corte escogidas.

Una vez guardados los filtros necesarios en una variable tipo *cell array*, y formateada la información referente a notas, se puede empezar a procesar el audio. Se realiza un primer ajuste de inicios y duraciones del *array* guardado en función de las duraciones de los archivos introducidos.

Se trata de comparar las duraciones del audio (ya delimitado en función de inicio y final del sonido) con la duración de la nota final datada en el array. Utilizando el cociente entre ambas se redimensionan los valores para hacer coincidir los minutajes.

Este ajuste nos asegura que los valores de inicio y las duraciones van a ser relativamente acertados, y evita errores al enventanar la señal de audio en busca de notas.

En este momento, ya se tienen las condiciones adecuadas para comenzar a buscar por nota; para ello, se recorre en un bucle todos los índices del array.

Para cada nota, será necesario establecer el intervalo de muestras donde es más probable que se encuentre. Este intervalo de muestras será filtrado, gracias al determinado filtro del banco anteriormente creado, y después sometido al algoritmo de búsqueda.

En versiones anteriores, se encogía un intervalo único, en el cual se aplicaba la búsqueda. Probando el método, se establece como principio, el uso de dos niveles de enventanado, esto resulta más eficiente y estable debido a varios motivos explicados más adelante.

Los márgenes para escoger el intervalo de audio dependen directamente de los valores del array de notas. Dado que estos son orientativos, y que la interpretación musical es humana, y por tanto inexacta, el algoritmo de búsqueda no siempre se encuentra con una nota definida dentro del intervalo escogido.

Al observar esta condición, se decide implementar la posibilidad de ensanchamiento de la ventana, para ello, se necesita que la parte filtrada sea mayor que la destinada a búsqueda de notas. De otra forma, volveríamos a filtrar datos ya obtenidos.

Es por ello que se separan dos tipos de límites para intervalos, por un lado límites de filtrado, y por otro límites de búsqueda. Ambos dependerán de los valores presentes en el array. Los márgenes tomados en ambos casos se fijan de forma experimental, observando el funcionamiento del algoritmo. Se establece un compromiso entre la posibilidad de error y la de encontrar notas que no son las buscadas.

Por otro lado, es necesario establecer límites en cuanto a la amplitud de la nota buscada, es decir, un umbral que determine si los valores encontrados en el intervalo de búsqueda son suficientes. Se establece la condición directamente en la función de búsqueda, de forma que esta devuelva nota siempre y cuando la oscilación detectada tenga amplitud suficiente.

El límite impuesto dependerá de una media extraída del fragmento ya filtrado. Se eliminan las partes de valor cercano a cero, para quedarse con los conjuntos que probablemente serán notas, y de estos se obtiene una media aritmética. Este es el valor que se le pasa a la función de búsqueda.

La idea que rige el método es la posibilidad de ensanchamiento de la ventana en caso de que se encuentre la nota parcialmente, o simplemente no se encuentre nota.

La ventana se ensanchará unilateralmente o bilateralmente en función de los resultados de la búsqueda. Se distinguen tres posibilidades.

- Nota empezada (note_init = 0) → la ventana se ensancha hacia el inicio del audio
- Nota sin acabar (note_fin = 0) → la ventana se ensancha hacia el final del audio
- Nota no encontrada (note_init = note_fin = 0) → la ventana se ensancha hacia ambos lados

En todo caso, existe la posibilidad de que el algoritmo no encuentre la nota en el fragmento de filtrado, o que encuentre notas adyacentes del mismo tono.

Para limitar en el segundo de los casos, se elimina la parte correspondiente a la nota anterior (ya conocida) para evitar errores. Igualmente se dan ciertos errores por diferencia de amplitud entre notas.

Será necesaria, además de la búsqueda de notas, un proceso de reajuste de tiempos en función de los datos hallados experimentalmente. Este se hace, igualmente, nota a nota, comparando el tiempo detectado con el esperado, y dimensionando los tiempos consecutivos en función de la relación entre ambos. Se realiza, entonces, solo en caso de encontrar la nota.

La parte crítica de este proceso radica en el margen de variación permitida por el algoritmo. Un margen pequeño causara que el proceso sea poco sensible a cambios bruscos de BPM, o notas interpretadas con libertad. Por otro lado, un margen demasiado grande dejará pasar errores de la función de búsqueda como desplazamientos grandes en la señal de audio, y producirá errores en el array final.

Para cada nota encontrada, siempre y cuando el inicio detectado y el teórico no difieran más de lo permitido, se realiza tanto la resta como el cociente de los valores.

Se utiliza la diferencia entra ambas para retrasar o adelantar los tiempos sucesivos. Esta es la parte de reacción rápida a las diferencias encontradas. Por otro lado, y de forma menos frecuente, se valora la posibilidad de que el tempo haya variado. Este cálculo depende del cociente de resultados experimental y teórico; si en sucesivas muestras resulta ser de magnitud diferente, se establece que el tiempo ha variado, y por tanto las duraciones de las notas también se tienen que adaptar, no sólo la localización.

Se utiliza el mismo cociente para realizar dicho cambio, aplicándolo sobre tiempos de inicio y duraciones por igual.

Salidas

La salida de este script es el array de datos corregido, es decir, sincronizado con la interpretación introducida.

Mejoras

La principal mejora a implementar para este módulo es el uso de más información proveniente del archivo introducido (siempre que este sea MXML).

DIVISIÓN Y ANÁLISIS DE AUDIO

Descripción y funcionalidades

Este módulo se encarga del análisis de la pista de audio, así como de su fragmentación para incluirla en el vídeo generado más adelante. Esta división dependerá de la frecuencia de muestreo del archivo de audio introducido, así como de los fotogramas por segundos escogidos para el vídeo de salida del sistema.

En cuanto al análisis, se limita a realizar un sencillo procesamiento para extraer la intensidad musical por cada fragmento. Esta parte es una de las probables ampliaciones del sistema.

Entradas

Como entrada se tendrá tanto la variable en la que se almacena el archivo de audio introducido como los fotogramas por segundo (fps) escogidos para el vídeo.

Procesamiento

La lógica de esta parte es la más simple y reducida. Se trata de un bucle que separa el archivo de audio en función de dos variables calculadas o determinadas previamente.

Utilizando el fps (fotogramas por segundo) del vídeo final, se calcula el tiempo de cada fotograma, y por tanto del fragmento de audio que le corresponderá.

$$t_{fragmento} = 1/fps$$

Dado un tiempo de audio para cada fragmento, y conocida la frecuencia de muestreo del archivo, se extrae el número de muestras por fragmento multiplicando ambos valores.

$$muestras_{fragmento} = t_{fragmento} * f_s$$

Por ejemplo, para un vídeo con $fps = 25$, un valor muy común en varios formatos de vídeo, y un archivo de audio de $f_s = 44100\ Hz$, un estándar de calidad de audio, obtendremos un valor de:

$$muestras_{fragmento} = 0.04\ seg * 44100\ Hz = 1764\ muestras$$

Una vez se tiene el número de muestras que debe tener cada fragmento, se puede calcular la cantidad de fragmentos en los que dividir el archivo de audio de entrada. Este será el redondeo del cociente entre la longitud total en muestras y las muestras por fragmento.

Con estos datos recorreremos un bucle que vaya guardando en un array las determinadas muestras de cada fragmento. Al mismo tiempo, se aprovecha para calcular la energía de cada fragmento. Simplemente se realiza un sumatorio de las muestras al cuadrado para cada iteración, guardando el resultado en un vector.

Salidas

La salida del módulo serán los fragmentos de audio, guardados en un array, así como el vector de amplitudes relacionado con éste.

Mejoras

La principal mejora a tener en cuenta en referencia a este módulo será el análisis de más parámetros de la señal de audio. Estos parámetros estarán relacionados con recursos musicales típicos. Esta mejora está directamente relacionada con el aprovechamiento de la información otorgada por los archivos MXML. En estos podremos encontrar señalizaciones de los posibles recursos utilizados por el/los interprete/s. Esto ayudará en su localización práctica.

Por ejemplo, se encontrarán variaciones rápidas de amplitud en los fragmentos en lo que el artista interprete un trémolo. Evidentemente, esto no coincidirá estrictamente con la partitura, dada la libre interpretación de la obra, pero en la mayoría de los casos nos dará una aproximación de su localización. De la misma forma encontraremos notas ligadas, vibrattos, y demás recursos musicales.

Además de estos componentes de la partitura, el sistema podría ser capaz de representar el paso de nota a nota con mayor fidelidad si se detecta la variación de frecuencia entre las mismas. Claro está, en instrumentos de estas características.

CREACIÓN DE VÍDEO

Descripción y funcionalidades

Este módulo es el encargado de, mediante los datos obtenidos anteriormente, generar una señal de video y audio sincronizada que sea representativa de la musicalidad de la pista de audio introducida. Al realizarse, como el resto del proyecto, en la plataforma Matlab, se encuentras serias limitaciones gráficas que complican el trabajo. Sin embargo, dado que el objetivo de este proyecto no era sino conseguir módulos eficientes que sirvan de base para futuras mejoras, no resulta preocupante que el vídeo generado sea de carácter sencillo.

Entradas

Este módulo tendrá como entradas, por un lado, el array de notas y posiciones temporales corregido. Este se habrá generado a partir de la información de los archivos introducidos, y sincronizado gracias al módulo anterior.

Por otro lado, se le pasan los fragmentos de audio relativos a cada fotograma del vídeo. Estos dependerán de la frecuencia de muestreo de la señal de audio introducida y de los fotogramas por segundo que se escojan para el vídeo de salida. Además, se le pasa un vector de amplitudes relativo a la sonoridad de la pista en cada uno de los fragmentos. En versiones posteriores se le introducirán más datos con el fin de generar un video más complejo y realista.

Además de lo citado, serán necesarias algunas características que definan el vídeo salida del módulo. Una de ellas será los fotogramas por segundo deseados en el vídeo a generar. Por otra parte, las dimensiones propias del mismo, expresadas en altura y anchura de la imagen.

Procesamiento

Una vez contextualizado el marco de este módulo, se explicará su proceso de creación, así como el procedimiento que implementa. Como generalidad, se establece que la representación con desplazamiento de las notas hacia la izquierda, dado que resulta más natural para el usuario final.

Para simplificar el script, éstas se representarán mediante círculos en movimiento. En el momento de interpretación de una determinada nota, se dibujará encima de un circulo ya existente otro de diferente color y tamaño. Es método será suficiente para comprobar la calidad de la sincronización realizada por el módulo previo.

Como alternativa tendremos otra versión que dibuje encima de los círculos rectángulos. Estos serán de altura constante y longitud variable y dependiente de la duración encontrada para cada nota. El motivo de realizar esta segunda versión radica en la depuración y mejora del algoritmo de búsqueda. Mediante esta variante podremos observar cuantas de las notas halladas se asemejan en duración a la real.

Resulta fácil deducir que el eje horizontal de la imagen corresponde al tiempo de reproducción. No es tan evidente, sin embargo, el significado práctico del eje vertical. Estará directamente relacionado con la altura de cada nota, entendiendo altura como frecuencia de las mismas.

La imagen será sobre fondo negro, y se dibujará, de forma que se mantenga inamovible una línea vertical situada en la parte central del cuadro. Esta corresponderá con el punto de referencia de sincronía entre audio y video. Será en el instante en el suene cada determinada nota cuando intersecten con esta línea de referencia. En ese punto se dibuja el circulo o rectángulo superior.

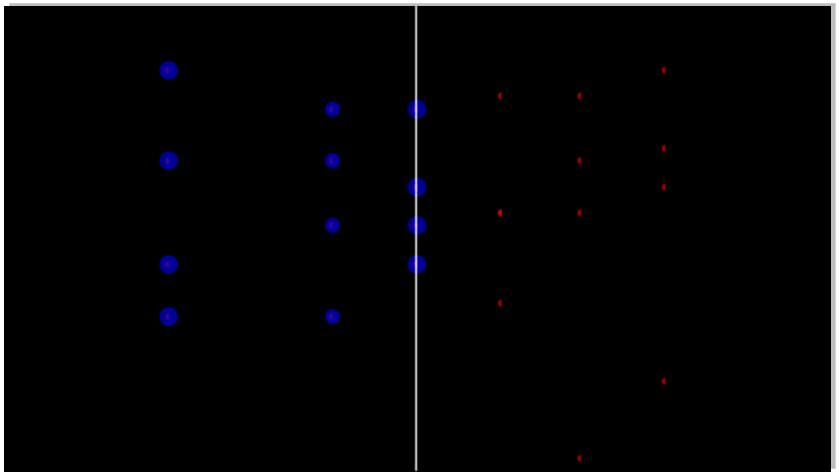


Figura 19

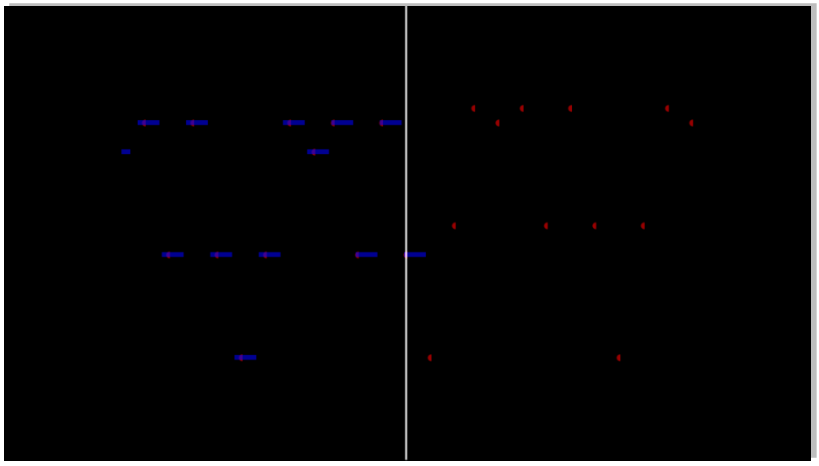


Figura 20

Por un lado, se crearán círculos rojos en el extremo izquierdo del cuadro, una vez dibujados el array que contiene la imagen se desplaza cada (1/fps) 6 pixeles a la derecha (para una anchura especificada de 720, en otro caso variará). Cuando estos lleguen a la línea central, se dibujará encima de cada uno de ellos otro circulo azul, para mostrar que la nota acaba de ser tocada; estos círculos serán de tamaño proporcional a la energía de la señal en ese instante.

Uno de las acciones previas al dibujado, será escalar las frecuencias obtenidas entre 0 y 1, de forma que las podamos usar como coeficiente de altitud de la nota, multiplicándolo por la altura del cuadro. Por otro lado, habrá que crear el objeto de vídeo donde iremos escribiendo cada fotograma. Esto se hace mediante la clase vidObj de Matlab. Este nos permite tanto elegir el archivo de salida, como su formato, fps y posibilita la entrada de audio por cada fotograma.

Una vez tenemos creado el objeto de vídeo, pasamos a dibujar sobre el cuadro de imagen. La primera parte del vídeo será en silencio, esperando a que las notas lleguen desde el extremo izquierdo a la línea central. Esta parte será un bucle separado, en el que se introducirán tramas de ceros como audio. Además, en este caso específico solo será necesario representar los círculos rojos, correspondientes a las notas por tocar.

En el siguiente bucle se empieza a representar desde que la primera nota llega a la línea de ejecución

Mejoras

Una de las futuras mejoras será, sin duda, el uso de otro motor gráfico externo a Matlab, parte de otro u otros lenguajes de programación.

Así mismo, y como se ha comentado en apartados previos, se prevé la extracción de más información en caso de la utilización de formato MXML. Mediante esta, y las capacidades más avanzadas aportadas por otro tipo de lenguajes de programación, será posible aproximar el resultado a los vídeos de referencia ya citados.

ANEXO 1: GUÍA DE USO

En este anexo se dan unas sencillas explicaciones sobre la ejecución de los scripts contenidos en la carpeta anexa. Estas indicaciones podrán encontrarse igualmente en el archivo txt llamado readme e incluido en dicha carpeta.

En la carpeta incluida en el proyecto, se encontrarán todas las funciones necesarias para ejecutar el sistema, así como se encontrará una carpeta con archivos de prueba y otra que contendrá el vídeo una vez terminado el script.

Situándonos en la carpeta principal, ejecutamos la función 'main'. Al hacerlo, nos aparecerá una ventana emergente para escoger el archivo de información de pista. Solo nos aparecerán los archivos compatibles con el sistema (MIDI o XML*). Una vez escogido un archivo, nos aparece otra ventana para el audio correspondiente. Cuando el sistema tiene los dos archivos, analiza de forma preventiva su compatibilidad. Se nos dará una respuesta de compatibilidad; esta es orientativa*. En caso de compatibilidad, el algoritmo sigue ejecutándose, en el caso contrario, nos pregunta si se quiere continuar.

Por la ventana de comandos se nos irá indicando el estado del proceso. Al terminar, podremos encontrar el vídeo de salida en la carpeta 'Output'.

*El archivo MXL descargado deberá descomprimirse, el sistema sólo acepta formato XML.

*Se puede dar incompatibilidad si no se encuentran la mayoría de las notas o si las longitudes son muy dispares.

ANEXO 2: FORMATOS MIDI Y MXML

En este anexo se detallan las principales características de los formatos relativos a información de pista utilizados en el proyecto.

MIDI

Las siglas MIDI son una abreviatura de Musical Instrument Digital Interface. Se trata de un protocolo de comunicación que apareció en el año 1982, fecha en la que distintos fabricantes de instrumentos musicales electrónicos se pusieron de acuerdo en su implementación. Aunque originalmente se concibió como un medio para poder interconectar distintos sintetizadores, el protocolo MIDI se utiliza actualmente en una gran variedad de aplicaciones: grabación musical, cine, TV, ordenadores domésticos, presentaciones multimedia, etc.

Dado que este protocolo es bastante eficiente en cuanto a enviar cantidades de datos relativamente grandes a una velocidad respetable, se ha convertido en un elemento de gran utilidad para compositores, educadores, programadores y todo tipo de músicos que necesitan un protocolo que comunique sus diferentes aparatos. Con la ayuda de un ordenador o un secuenciador hardware, permite crear arreglos multipista, líneas o partes instrumentales.

Veamos de forma un poco más precisa algunas de las ventajas que proporciona.

Generar sonido a partir de un sintetizador MIDI en vez de hacerlo partiendo de un sampler tiene algunas ventajas. La primera de ellas es que se necesita una gran cantidad de espacio de almacenamiento para guardar el audio muestreado (p.ej., en forma de archivos .WAV o AIFF).

Se necesitan unos 10 Mb de espacio en disco para almacenar 1 minuto de audio estéreo muestreado en calidad CD (16 bits y 44,1kHz). En comparación, los archivos de datos MIDI tienen un tamaño insignificante. Una secuencia MIDI típica utiliza sólo unos 10 Kb por minuto.

El funcionamiento del protocolo se basa en que el archivo MIDI no contiene datos de audio muestreado, sino más bien una serie de instrucciones que el sintetizador u otro generador de sonido utiliza para reproducir el sonido en tiempo real.

Estas instrucciones son mensajes MIDI que indican al instrumento qué sonidos hay que utilizar, qué notas hay que tocar, el volumen de cada una de ellas, etc.

El reducido tamaño de estos archivos implica que un ordenador poco potente puede hacerse cargo de complejos arreglos musicales sin ni siquiera inmutarse. En cambio, puede llegar a ser imposible que una de estas máquinas sea capaz ni tan siquiera de reproducir unas pocas pistas de audio muestreado en calidad CD.

Otras ventajas:

- Ofrece la posibilidad de editar la música con facilidad.
- Permite alterar la velocidad de reproducción y la altura tonal de los sonidos de forma independiente.

Al respecto de este último punto, al cambiar la velocidad de una grabación en cinta, CD, disco duro, etc., cambiará la altura tonal del sonido.

MusicXML

Se trata de un formato abierto de notación musical basado en XML (eXtensible Markup Language), un lenguaje de marcas Extensible.

Fue desarrollado por Recordare LLC, derivando varios conceptos clave de formatos académicos existentes (como el Musedata de Walter Hewlett y el Humdrum de David Huron). Fue diseñado para el intercambio de partituras, particularmente entre diferentes tipos de software editor de partitura.

Es capaz, mediante etiquetas, de definir todas las características de referentes a una partitura corriente. Es decir, encontraremos etiquetas para cada posible propiedad expresada en la partitura, como la división del compás '<**time**>', la clave '<**clef**>', o las diferentes partes de la partitura '<**score-part**>'. Para cada una de estas etiquetas se establecen ciertos atributos propios de la característica determinada. De esta forma, para una nota, tendremos especificadas etiquetas como el tono '<**pitch**>', o su duración'<**duration**>'.

Además de esta información referenciada a las características musicales expresadas en la partitura, encontraremos otras etiquetas cuya función sea de tipo organizativa, o referente a la representación de cada elemento. También algunas que definen las características de los márgenes de la partitura.

En definitiva, este lenguaje, junto con el software adecuado, nos permite codificar, enviar, leer y modificar partituras digitales.

ANEXO 3: COMPARATIVA MÉTODOS FILTRADO

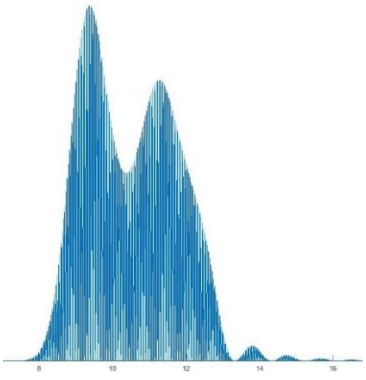
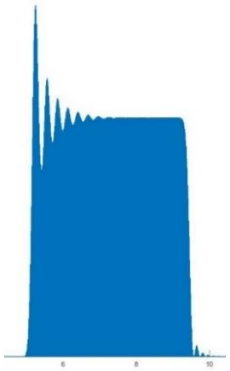
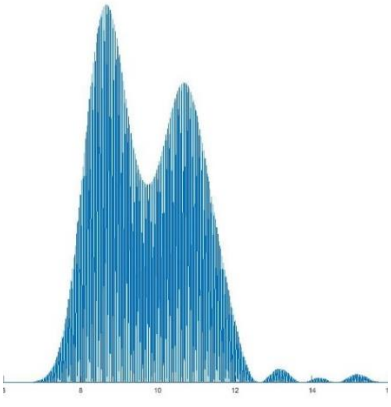
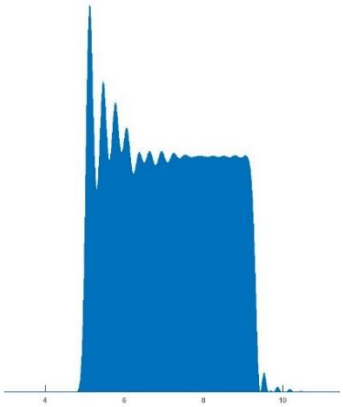
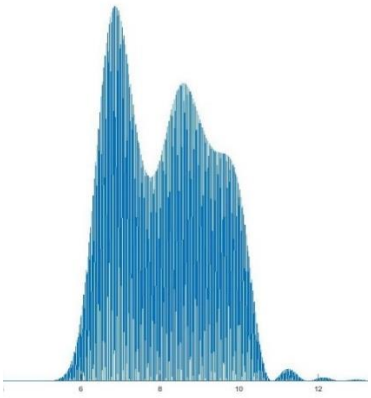
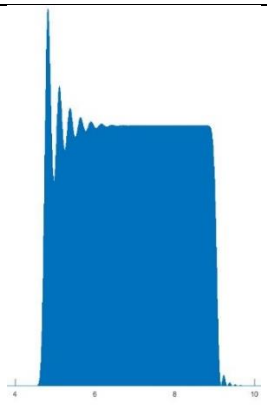
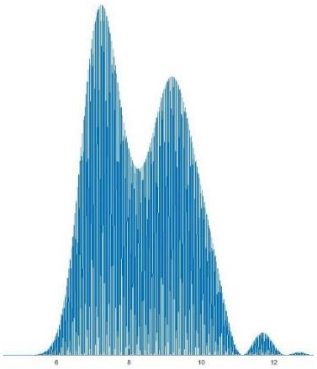
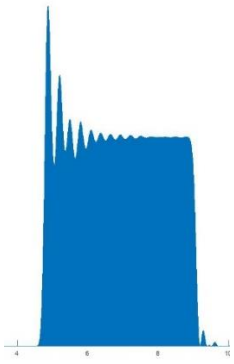
Método/Nota	C2	G4
Butterworth		
Chebychef 1		
Chebychef 2		
Elliptic		

Tabla 4

ANEXO 4: FUNCIONES

create_filters

Entrada

- Rango de notas (en codificación MIDI)
- Frecuencia de muestreo

Salida

- Cell array de filtros

Esta función se vale de la herramienta 'designfilt' de Matlab para generar el banco de filtros necesarios. En concreto utiliza la versión que genera el filtro paso banda a partir de frecuencias de paso y de rechazo, así como las atenuaciones de las bandas de rechazo y el máximo rizado en la de paso. En el siguiente esquema podemos ver a que hace referencia cada valor.

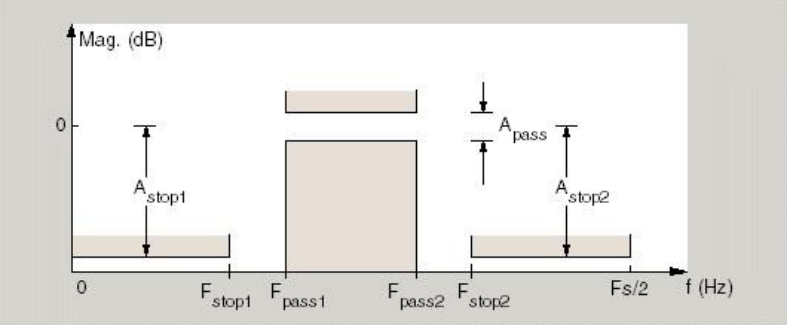


Figura 21

Para obtener dichos valores, partimos de la frecuencia en codificación MIDI y del valor de referencia de 440Hz del 'La' central. Se pasa de dicha codificación a las frecuencias en Hz, y con las mismas se generan las especificaciones para cada filtro. Las bandas de paso estarán comprendidas entre:

$$f_{pass_1} = \frac{f_c}{K'}$$

$$f_{pass_2} = f_c * K'$$

$$con \quad K' = 2^{1/24}$$

Por otro lado, las frecuencias para definir las bandas de rechazo serán:

$$f_{pass\_1} = f_c/K$$

$$f_{pass\_2} = f_c * K$$

Así mismo, se escogen atenuaciones de rechazo de 100dB, y un máximo rizado de 1dB. El proceso de diseño nos deja elegir el tipo de aproximación matemática usada para crear el filtro. Mediante la herramienta DesignMethods se nos dan los métodos disponibles para cada caso.

Para escoger el más apropiado, se hace una comparativa basada en el retardo añadido, así como el rizado introducido. Se eligen dos frecuencias probables y distantes para testear cada uno de los métodos. Por un lado, una frecuencia relativamente baja, y por otro una relativamente alta. Al decir relativamente, se trata de destacar el hecho

de que la altura relativa de la nota es totalmente dependiente de la pieza musical, así como del instrumento o instrumentos presentes en la misma. Se descartan, a priori, dos de los posibles métodos. La decisión radica en los tiempos de ejecución de estos, excesivamente altos.

En las siguientes figuras podemos ver la señal filtrada para realizar la prueba, así como los resultados resumidos en una tabla:

Aunque no se perciben diferencias muy significativas, se puede observar un menor rizado, así como una mayor fiabilidad en cuanto a la amplitud original de la señal en el caso del método Chebyshev tipo 2. Este será, por tanto, el método escogido.

init_filters

Entrada

- Rango de notas (en codificación MIDI)
- Frecuencia de muestreo

Salida

- Cell array de filtros
- Correcciones de retardo
- Correcciones de anchura

Esta función comenzará llamando a la anteriormente mencionada 'create_filters' y se ocupará, una vez creado el cell array de filtros, de caracterizarlos. Dado el rizado de estos filtros, así como el transitorio de los mismos, la señal filtrada puede resultar seriamente variada y, a causa de efectos del filtrado y de la función de búsqueda, desplazada en el tiempo. Será necesario definir para cada uno de ellos unos parámetros que nos ayuden a contrarrestar en la medida de lo posible esta propiedad de no idealidad tanto del filtrado como de la búsqueda de notas.

Para ello se crea para cada frecuencia del rango necesario una señal de entrada, de amplitud y duración conocidas. Después se pasa esta por su determinado filtro, y se extrapolan datos comparando el resultado del filtrado con la original. Se necesita saber el retardo introducido, así como el posible escalado sufrido por la señal.

Para conocer esta información sobre la señal obtenida, recurrimos a la función 'find_note', que busca el inicio y final de una determinada nota. Más adelante se explicará con detenimiento el funcionamiento de este algoritmo.

Una vez se obtienen los valores de inicio y final de la nota filtrada, se relaciona con los valores conocidos en el caso de cada frecuencia. Se rellena el vector 'delay' con los retardos introducidos por cada filtro y el vector 'dur_correction' con el inverso del escalado de la duración de la nota filtrada.

El siguiente paso es escalar los valores de tiempos para hacerlos coincidir con el audio. Ésta será la primera aproximación al sincronismo. Tomamos como referencia de la variable de datos el tiempo de la última nota incluida, que se nos da en segundos. En cuanto al audio, sólo hay que dividir el número de muestras entre la frecuencia de muestreo, previamente guardada en la variable Fs. Con la relación entre ambos datos, se re-escalan las duraciones de las notas. Después se vuelve a hacer la suma iterativa de inicio de cada nota y su duración correspondiente.

Una vez realizado este proceso, se pasa al procesado de la primera nota, con el objetivo de fijar el comienzo real de la pieza sonora. Se busca la primera frecuencia en la información obtenida del MIDI o MXML y se escoge una parte lo suficientemente grande para contenerla en su totalidad. Una vez se tiene la señal de entrada, se aplica el algoritmo de búsqueda de nota.

find_note

Entrada

- Señal audio filtrada y elevada al cuadrado
- Umbral de nota

Salida

- Inicio de la nota
- Final de la nota

Para explicar el funcionamiento del algoritmo nos serviremos de un caso práctico referente a una nota del audio de prueba. En la siguiente figura podemos ver el extracto de audio introducido en la función, este ya ha sido filtrado y elevado al cuadrado, luego representa la energía de la señal en un rango de frecuencias restringida, para ese intervalo:

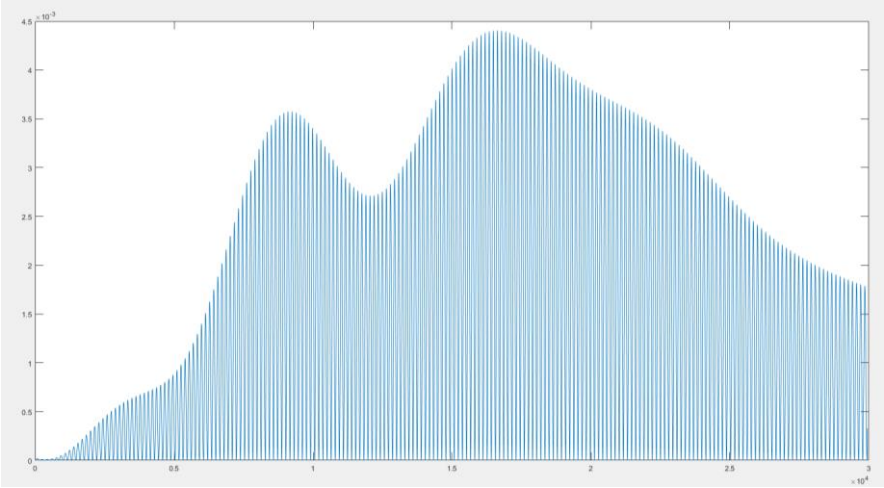


Figura 22

El cometido del algoritmo es encontrar el inicio y final de la nota contenida en el audio. Para ello, será necesario extraer la envolvente de la señal, de lo contrario, las continuas oscilaciones a la frecuencia de la nota estudiada nos impedirían encontrar determinados puntos de energía.

Utilizando la función 'envolve' del toolbox … de Matlab extraemos dicha envolvente.

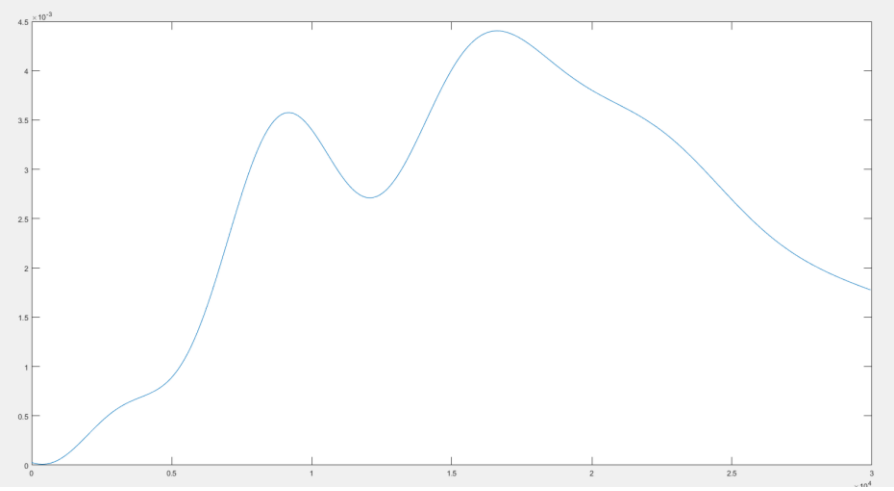


Figura 23

Una vez tenemos la energía de la señal representada, y suponiendo que estamos escogiendo el tramo de nota correctamente, establecemos un threshold en función del máximo de energía encontrado. En concreto se usa la mitad del mismo, como queda representado en la figura 20.

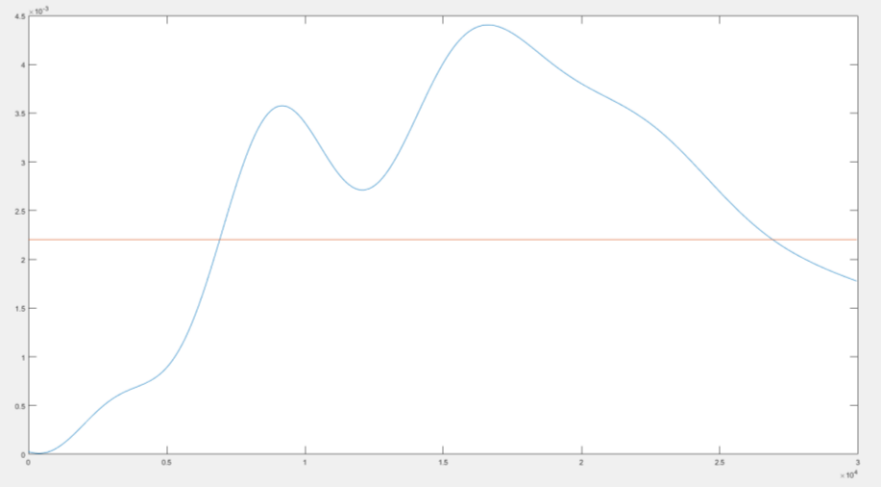


Figura 24

Se utiliza la función 'find' para encontrar los valores que sobrepasan este umbral. En el mejor de los casos, se encontrará un solo conjunto relativamente centrado, esto significará que la ventana escogida contiene completamente a la nota. En este caso, el inicio y final del conjunto de valores mayores que el umbral coincidirán con el inicio y final de la nota en cuestión.

Otra posibilidad es que el conjunto encontrado no este centrado, es decir, que susu valores inicial o final coincidan con los del fragmento respectivamente. Estos casos se corresponden con una nota empezada o no acabada, y podrá tener causas varias:

- Cambio de BPM
- Interpretación del artista
- Errores en reajustes de tiempos

En el caso de que la nota esté empezada, se pondrá la variable referente al inicio de la nota a cero, como indicativo de error en el algoritmo. En el contrario, nota no acabada, se actuará de forma inversa, guardando el inicio de la nota e igualando el final a cero. Será el script superior el que reaccione a este error.

Estos valores devueltos por la función repercutirán en el procesamiento del algoritmo de sincronismo, ensanchando este el fragmento de audio escogido.

Un último caso a considerar, y sin duda el más conflictivo, será encontrar más de un conjunto de valores mayores al umbral. Esto puede ser debido a varias causas como notas del mismo tono adyacentes, sobredimensionamiento de la ventana..etc.

El proceder del algoritmo separa dos casos determinados. Por un lado, si encuentra más de dos conjuntos de muestras por encima del umbral, se queda con el centrado en la ventana, suponiendo que los otros dos son reductos de otras notas al inicio y final del fragmento.

En el caso de dos conjuntos, se revisa si alguno abarca la primera muestra del fragmento de audio, esto significa que, de ser la nota buscada, esta estaría empezada. Si se da ese caso, establece un mínimo para el final de ese conjunto. Si es mayor a la mitad del fragmento, se toma como nota empezada, si no lo es, se toma como nota el segundo conjunto.

Si se da el caso de que ningún conjunto empieza en la primera muestra, el algoritmo trabaja con los tamaños de estos, comparando ambos ponderadamente. Se establece prioridad al primero, ya que experimentalmente es el caso más probable.

ANEXO 5: TEORÍA MUSICAL

En este anexo se introducen algunos conceptos básicos sobre las relaciones matemáticas de la escala musical.

Octava: sucesión de las ocho notas de la escala musical

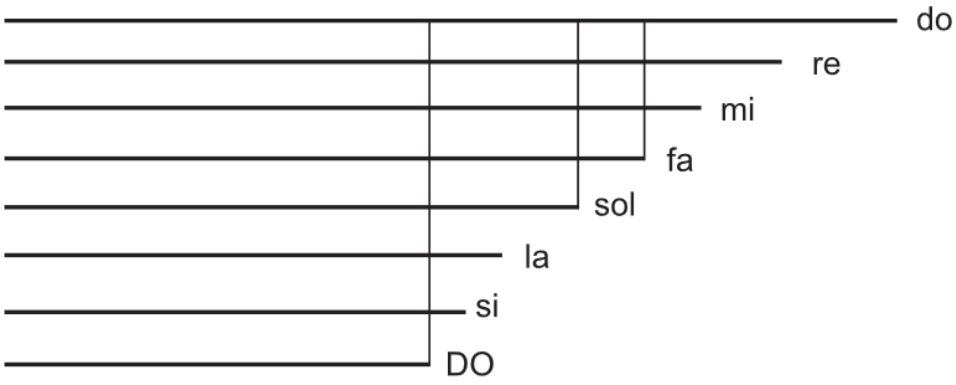


Figura 25

Cada una de estas notas tiene sus correspondientes octavas superior e inferior. Por ejemplo, la nota **DO** es la octava superior de la nota **do**. La nota tomada como base de la escala (en este caso la nota **do**) se denomina tónica.

Notas musicales: al iniciarse el estudio matemático de la escala musical, era común la utilización de longitudes de cuerdas vibrantes para caracterizar las notas musicales. La relación entre estas longitudes y las notas de la escala se denominaba **monocardio.** Hoy en día las notas musicales no se definen a partir de la longitud del objeto vibrante, sino a partir de la frecuencia de vibración de la onda sonora emitida por dicho objeto. La frecuencia y la longitud de una onda sonora se encuentran vinculadas por medio de la ecuación:

$$f = \frac{v}{l}$$

Donde **f** es la frecuencia en Hertz; **v** es la velocidad del sonido en m/seg y **l** es la longidus de onda en metros. Las bajas frecuencias corresponden a notas graves, las altas, caracterizan a los tonos agudos.

En la siguiente imagen podemos observar las fracuencias relacionadas con una octava musical determinada, comenzando por **do.**

Media aritmética: es una de las primeras relaciones de proporción entre notas citada en la historia. Se rige por la siguiente fórmula:

$$b = \frac{a + c}{2}$$

Reemplazando **a** y **c** por las notas por los extremos de la octava de **do**, obtendremos el siguiente resultado:

$$b = \frac{261 + 522}{2} \cong 391.5$$

Es aproximadamente el valor de la frecuencia de la nota **sol**, es decir, la media aritmética determina la relación entre **do** y **sol**, o, lo que es lo mismo, entre **do** y su **octava**. El cociente entre dichas notas es conocido como intervalo de quinta.

$$\frac{391.5}{261} \cong \frac{3}{2}$$

Media armónica: se expresa de la siguiente forma

$$b = \frac{2ac}{a+c}$$

Reemplazando **a** y **c** respectivamente por las notas **do** y **DO** obtenemos:

$$b = \frac{2 * 261 * 522}{261 + 522} = 348$$

Que corresponde aproximadamente con la nota **fa.** La proporción armónica define la relación entre las notas **do** y **fa**, es decir, la relación entre **do** y su **cuarta**. Esta relación se denomina cuarta o cuarta perfecta, y su valor se puede obtener a partir del cociente entre ambas.

$$\frac{348.3}{261} \cong \frac{4}{3}$$

Media geométrica: expresada algebraicamente como

$$\frac{a}{b} = \frac{b}{c}$$

Caracteriza la relación entre octavas sucesivas, siendo **a** y **b** las frecuencias extremo de una de las octavas. Entonces serán **b** y **c** las de la octava consecutiva. El cociente entre ambas se denomina **intervalo de octava**.

$$\frac{522}{261} \cong \frac{2}{1}$$

Relación entre tonos: la relación entre notas consecutivas viene dada por el cociente entre medias aritmética y armónica.

$$\frac{\frac{a+c}{2}}{\frac{2ac}{a+c}}$$

Remplazando **a** y **c** por las frecuencias extremo de una octava, se obtiene lo siguiente.

$$\frac{\frac{216+522}{2}}{\frac{2*216*522}{216+522}} \cong \frac{9}{8}$$

Este es el valor de un **intervalo de tono**, es decir, el factor que relaciona notas consecutivas de la escala musical.

Tonos y semitonos: Las ocho notas de la octava se encuentran separadas entre sí por intervalos de tono o de semitono. Según se ha visto, el valor del intervalo de tono es 9/8. Sin embargo, y a pesar de su nombre, el intervalo de semitono no equivale, desde el punto de vista matemático, a la mitad de un intervalo de tono, puesto que está definido por la fracción

$$\frac{256}{243}$$

La octava está compuesta por cinco tonos y dos semitonos, distribuidos de la siguiente manera:

$$do - re - mi \cap fa - sol - la - si \cap do$$

Donde '–' denota un tono (T) y '∩' un semitono (S). Se cumplen las siguientes relaciones:

Intervalo de quinta → 3T y 1S

Intervalo de cuarta perfecta → 2T y 1S

Intervalo de cuarta aumentada o tritono → 3T

La escala templada o temperada: En el siglo XVIII Juan Sebastián Bach (1685- 1750), en su obra '*El clave bien temperado*', implementó ciertas modificaciones fundamentales que habían sido introducidas a la escala musical. Desde entonces, la escala corrientemente empleada en Occidente es la escala temperada.

En esta escala, existen once frecuencias intermedias entre una nota y su octava superior. Las doce frecuencias de la escala temperada se denotan

$$do - do\# - re - re\# - mi - fa - fa\# - sol - sol\# - la - la\# - si$$

donde el signo # indica una nota "sostenida". En el piano, estas doce frecuencias se corresponden con las sucesivas series de siete teclas blancas y cinco teclas negras.

Según se observa en el esquema anterior, en la escala temperada se intercalan notas (las notas #) entre aquellos sonidos de la octava que distan entre sí en un intervalo de un tono. Como resultado de estas modificaciones, todos los sonidos sucesivos de la escala temperada están separados entre sí por una distancia de un semitono. En otras palabras, entre dos notas consecutivas cualquiera de la escala temperada existe siempre, exactamente, el mismo intervalo. En la escala temperada las frecuencias f_n y f_{n+1} de dos notas sucesivas verifican la siguiente relación:

$$f_{n+1} = f_n * K$$

Donde $n = 0, 1, 2, \ldots$.

Esta ecuación caracteriza la relación entre las diferentes frecuencias de la escala templada. En esta expresión f_0 es la frecuencia de la nota menor o nota tónica; y la constante K tiene el valor 1,059.

El valor de K proviene del cociente de longitudes de onda entre notas de separación un semitono. El valor se puede obtener matemáticamente teniendo en cuenta la condición logarítmica de la escala musical. Esta se divide en 12 semitonos, y por tanto, la distancia entre ellos queda expresada como

$$\boldsymbol{K = 2^{1/12} = 1.059463 \cong 1.06}$$

En el proyecto se usará este valor para obtener las frecuencias de paso y de corte de los filtros.

REFERENCIAS

1. http://www.css-audiovisual.com/areas/guias/midi.htm (MIDI info)
2. http://repositori.uji.es/xmlui/bitstream/handle/10234/148045/TFM15_Salvador_Marti_Solsona.pdf?sequence=1 (OnSet detection info)
3. https://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/miditoolbox (MIDI Toolbox)
4. http://music.informatics.indiana.edu/code/musicxml/ (MusicXML Toolbox)
5. https://ccrma.stanford.edu/~jos/fp/Forward_Backward_Filtering.html (Filtrado bidireccional)
6. http://www2.dis.ulpgc.es/~li-pso/tema5/tema5.htm (Información sobre filtros)
7. http://ocw.uv.es/ingenieria-y-arquitectura/filtros-digitales/tema_2._revision_de_los_tipos_de_filtros_analogicos_mas_comunes.pdf (Información sobre filtros)
8. http://www4.tecnun.es/asignaturas/tratamiento%20digital/tema8.pdf (Información sobre filtros)
9. http://cp.literature.agilent.com/litweb/pdf/ads15/dfilter/df044.html (imágenes filtros)
10. http://www.palermo.edu/ingenieria/downloads/CyT6/6CyT%2003.pdf (teoría musical)
11. https://www.minidsp.com/applications/dsp-basics/fir-vs-iir-filtering (información filtros)
12. http://www.dtic.upf.edu/~egomez/teaching/sintesi/SPS1/Tema7-FiltrosDigitales.pdf (información filtros)